รายงานการทดลองที่ 3SA03

Parallel Programming With OpenMP

จัดทำโดย

นายปณิธาน ดวงขวัญ

รหัส 5735512036 Section 01

เสนอ

อาจารย์ฐิตินันท์ เกลี้ยงสุวรรณ

รหัสวิชา 242-301 Advanced Computer Engineering Laboratory I

คณะวิศวกรรมศาสตร์ ภาควิชาวิศวกรรมคอมพิวเตอร์

มหาวิทยาลัยสงขลานครินทร์

การทดลองที่ 3S A03

เรื่อง Parallel Programming with OpenMP

วัตถุประสงค์

เพื่อให้นักศึกษาเข้าใจหลักการพัฒนาโปรแกรมเพื่อการประมวลผลแบบขนานเบื้องต้นโดยใช้ OpenMP

อุปกรณ์การทดลอง

1. Dev C/C++ compiler with OpenMP
2. เครื่องคอมพิวเตอร์แบบ multi-core

งานและคำถามก่อนการทดลอง

อ่านเอกสารการทดลองและศึกษาข้อมูลเพิ่มเติมจากอินเตอร์เน็ต

1. Parallel Programming มีประโยชน์และมีความสำคัญอย่างไร เพราะเหตุใด
2. Thread กับ Process ต่างกันอย่างไร

คำถามก่อนการทดลอง

1. **Parallel Programming มีประโยชน์และมีความสำคัญอย่างไร เพราะเหตุใด**

   ตอบ โปรแกรมแบบขนานนั้นมีประโยชน์ในการทำงานในส่วนของโปรแกรมที่มีขนาดใหญ่และซับซ้อนเพราะการทำงานแบบขนานนั้นจะนำ thread หลายๆส่วนมาใช้งานเพื่อที่จะให้โปรแกรมทำงานได้เร็วและดีที่สุด โดย thread นั้นจะช่วยกันทำงานเพื่อให้โปรแกรมที่มีความซับซ้อนนั้นเสร็จได้อย่างมีประสิทธิภาพ

2. **Thread กับ Process ต่างกันอย่างไร**

   ตอบ **Thread** คือ หน่วยทำงานของ Process ซึ่ง Thread นั้นจะต้องรันอยู่ภายใต้ process ส่วน **Process** นั้นคือ โปรแกรมที่กำลังถูกประมวลผล หรือ กระบวนการทำงานของโปรแกรมใดโปรแกรมหนึ่ง

การทดลอง

คอมไพล์และรันโปรแกรมทั้ง 3 ตอนต่อไปนี้ ทั้งแบบ serial และ parallel แล้วตรวจสอบผลลัพธ์และจับเวลาการประมวลผลเปรียบเทียบกันโดยใช้ฟังก์ชันจับเวลาของ OpenMP บันทึกรายงาน และวิเคราะห์ผล ทั้งนี้ให้ดำเนินการดังนี้

1. ในตอนที่ 1 และ 3 ให้ปรับเปลี่ยนจำนวนเทรด (thread)
2. ในตอนที่ 2 และ 3 ผลของการคำนวณในแบบ parallel ต้องเหมือนกับแบบ serial แล้วอธิบายว่าเพราะเหตุใด
3. ในตอนที่ 3 ปรับเปลี่ยนขนาดของข้อมูล และจำนวนลูป (loop)
4. ในทุกตอน ให้สังเกตข้อมูล process และ thread ของ โปรแกรมในการทดลองที่กำลัง run อยู่ และดู performance ของ CPU บน Windows Task Manager

ตอนที่ 1 Hello world

Serial

hello world Serial.c

```c
#include <stdio.h>
int main(int argc, char *argv[])
{
 int t_id=1;
 double start,stop;
 start = omp_get_wtime();
 printf("Hello world from thread ID %d\n",t_id);
 stop = omp_get_wtime();
 printf("Time to compile this = %f",stop-start);
}
```

ผลการ RUN

```
G:\2560\lab advance I\Lab3\hello world Serial.exe
Hello world from thread ID 1
Time to compile this = 0.000000
--------------------------------
Process exited after 0.00925 seconds with return value 31
Press any key to continue . . .
```

Parallel

hello world Serial.c | hello world Parallel.c

```c
#include <omp.h>
#include <stdio.h>
int main(int argc, char *argv[])
{
 int t_id, num_t;
 double start,stop;
    start = omp_get_wtime();
 #pragma omp parallel num_threads(50)
 {
    num_t = omp_get_num_threads();
    t_id = omp_get_thread_num();
    printf("Hello world from thread ID %d/%d\n",t_id,num_t);
 }
    stop = omp_get_wtime();
    printf("Time to compile this = %f",stop-start);
}
```

hello world Serial.c | hello world Parallel.c

```c
#include <omp.h>
#include <stdio.h>
int main(int argc, char *argv[])
{
 int t_id, num_t;
 double start,stop;
    start = omp_get_wtime();
 #pragma omp parallel num_threads(100)
 {
    num_t = omp_get_num_threads();
    t_id = omp_get_thread_num();
    printf("Hello world from thread ID %d/%d\n",t_id,num_t);
 }
    stop = omp_get_wtime();
    printf("Time to compile this = %f",stop-start);
}
```

ผลการ RUN

G:\2560\lab advance I\Lab3\hello world Parallel.exe

```
Hello world from thread ID 24/50
Hello world from thread ID 25/50
Hello world from thread ID 26/50
Hello world from thread ID 27/50
Hello world from thread ID 28/50
Hello world from thread ID 29/50
Hello world from thread ID 30/50
Hello world from thread ID 31/50
Hello world from thread ID 32/50
Hello world from thread ID 33/50
Hello world from thread ID 34/50
Hello world from thread ID 35/50
Hello world from thread ID 36/50
Hello world from thread ID 37/50
Hello world from thread ID 38/50
Hello world from thread ID 39/50
Hello world from thread ID 41/50
Hello world from thread ID 40/50
Hello world from thread ID 42/50
Hello world from thread ID 43/50
Hello world from thread ID 44/50
Hello world from thread ID 45/50
Hello world from thread ID 46/50
Hello world from thread ID 47/50
Hello world from thread ID 2/50
Hello world from thread ID 0/50
Time to compile this = 0.011000
--------------------------------
Process exited after 0.01847 seconds with return value 31
Press any key to continue . . .
```

G:\2560\lab advance I\Lab3\hello world Parallel.exe

```
Hello world from thread ID 75/100
Hello world from thread ID 77/100
Hello world from thread ID 76/100
Hello world from thread ID 79/100
Hello world from thread ID 78/100
Hello world from thread ID 80/100
Hello world from thread ID 81/100
Hello world from thread ID 83/100
Hello world from thread ID 82/100
Hello world from thread ID 84/100
Hello world from thread ID 85/100
Hello world from thread ID 86/100
Hello world from thread ID 88/100
Hello world from thread ID 89/100
Hello world from thread ID 87/100
Hello world from thread ID 90/100
Hello world from thread ID 91/100
Hello world from thread ID 92/100
Hello world from thread ID 93/100
Hello world from thread ID 94/100
Hello world from thread ID 95/100
Hello world from thread ID 0/100
Hello world from thread ID 96/100
Hello world from thread ID 97/100
Hello world from thread ID 98/100
Hello world from thread ID 99/100
Time to compile this = 0.027000
--------------------------------
Process exited after 0.03481 seconds with return value 31
Press any key to continue . . .
```

สรุปผลการทดลอง

จากการทดลองจะพบว่าถ้าการทำงานแบบ Serial จะพบว่าคอมไพเลอร์จะใช้เวลาน้อยมากในการประมวลผลซึ่งต่างจากการทำงานแบบขนาน Parallel โดยการทำงานแบบขนาดสามารถใช้จำนวน thread ในการทำงานได้จากภาพประกอบด้านบนจะพบว่า ถ้าเรากำหนด thread น้อยจะทำให้เวลาในการประมวลน้อยลงเช่นกัน โดยจะสรุปได้ว่า ถ้าเรากำหนดจำนวน thread เยอะเวลาที่ใช้ในการประมวลผลก็จะเยอะตามขึ้นมา โดยสามารถดูได้จากผลการรันและภาพประกอบด้านบน

ตอนที่ 2 For loop

Serial

For loop Serial.c | hello world Serial.c | hello world Parallel.c

```
1  #include <stdio.h>
2  #include <math.h>
3  #define NUMBER 500000
4  int main(int argc, char *argv[])
5  {
6   int i, data[NUMBER];
7   double result=0.0,start,stop;
8   start = omp_get_wtime();;
9   for (i=0; i<NUMBER; i++)
10     data[i]=i*i;
11  for (i=0; i<NUMBER; i++)
12     result+=(sin(data[i])-cos(data[i]))/(sin(data[i])+cos(data[i]));
13  printf("Result = %f\n",result);
14  stop = omp_get_wtime();
15  printf("Time to compile this = %f",stop-start);
16 }
```

ผลการ RUN

```
G:\2560\lab advance I\Lab3\For loop Serial.exe
Result = 149987.885694
Time to compile this = 77.000000
--------------------------------
Process exited after 0.08617 seconds with return value 32
Press any key to continue . . .
```

Parallel (naïve)

For loop Parallel navie.c | For loop Serial.c | hello world Serial.c | hello world Parallel.c

```c
#include <omp.h>
#include <stdio.h>
#include <math.h>
#define NUMBER 500000
int main(int argc, char *argv[])
{
 int i, data[NUMBER];
 double result=0.0,start,stop;
 start = omp_get_wtime();
 for (i=0; i<NUMBER; i++)
    data[i]=i*i;
 #pragma omp parallel for
 for (i=0; i<NUMBER; i++)
    result+=(sin(data[i])-cos(data[i]))/(sin(data[i])+cos(data[i]));
 printf("Result = %f\n",result);
 stop = omp_get_wtime();
 printf("Time to compile this = %f",stop-start);
}
```

ผลการ RUN

```
G:\2560\lab advance I\Lab3\For loop Parallel navie.exe
Result = 132253.994983
Time to compile this = 0.029000
--------------------------------
Process exited after 0.03439 seconds with return value 31
Press any key to continue . . .
```

Parallel with critical section

For loop Parallel with critical section.c | For loop Parallel navie.c | For loop Serial.c | hello world Serial.c | hello world Parallel.c

```c
#include <omp.h>
#include <stdio.h>
#include <math.h>
#define NUMBER 500000
int main(int argc, char *argv[])
{
 int i, data[NUMBER];
 double result=0.0,start,stop;
 start = omp_get_wtime();
 for (i=0; i<NUMBER; i++)
    data[i]=i*i;
 #pragma omp parallel for
 for (i=0; i<NUMBER; i++)
 {
    #pragma omp critical
    result+=(sin(data[i])-cos(data[i]))/(sin(data[i])+cos(data[i]));
 }
 printf("Result = %f\n",result);
 stop = omp_get_wtime();
 printf("Time to compile this = %f",stop-start);
}
```

ผลการ RUN

```
G:\2560\lab advance I\Lab3\For loop Parallel with critical section.exe
Result = 149987.885694
Time to compile this = 1.078000
--------------------------------
Process exited after 1.087 seconds with return value 31
Press any key to continue . . .
```

Parallel with reduction

For loop Parallel with reduction.c | For loop Parallel with critical section.c | For loop Parallel navie.c | For loop Serial.c | hello world Serial.c | hel

```c
#include <omp.h>
#include <stdio.h>
#include <math.h>
#define NUMBER 500000
int main(int argc, char *argv[])
{
 int i, data[NUMBER];
 double result=0.0,start,stop;
 start = omp_get_wtime();
 for (i=0; i<NUMBER; i++)
    data[i]=i*i;
 #pragma omp parallel for reduction(+:result)
 for (i=0; i<NUMBER; i++)
    result+=(sin(data[i])-cos(data[i]))/(sin(data[i])+cos(data[i]));
 printf("Result = %f\n",result);
 stop = omp_get_wtime();
 printf("Time to compile this = %f",stop-start);
}
```

ผลการ RUN

```
G:\2560\lab advance I\Lab3\For loop Parallel with reduction.exe
Result = 149987.885694
Time to compile this = 0.056000
--------------------------------
Process exited after 0.06647 seconds with return value 31
Press any key to continue . . .
```

สรุปผลการทดลอง

ชนิดการประมวลผล	Result	Time(s)
Serial	149987.885694	0.056000
Parallel (navie)	86271.926770	0.038000
Parallel with critical section	149987.885694	0.887000
Parallel with reduction	149987.885694	0.041000

จากการทดลองจะพบว่าชนิดประมวลผลแต่ละชนิดจะได้ค่าที่ใกล้เคียงกันยกเว้น Parallel (navie) ที่จะพบว่าค่าทีได้ออกมานั้นแตกต่างจากการประมวลชนิดอื่นได้ชัดเจน ในส่วนของเวลาในการประมวลผลนั้นจะพบได้ว่าแบบ Parallel (navie) นั่นจะเร็วที่สุด ซึ่งที่เร็วรองลงมานั้น Parallel with reduction แต่ในส่วนของ Serial นั้นใช้เวลาในการประมวลผลนั้นปกติทั่วไป ส่วนในรูปแบบการประมวลผล critical section นั้นจะช้าที่สุดเพราะจะทำงานในส่วนของ Critical Section นั่นเองจึงทำให้ทำงานช้า

โปรแกรมทั้ง 3 รูปแบบนั้นประมวลผลได้ไม่ตรงกันโดยแบบ navie นั้นจะเป็นการทำงานโดยดูว่า thread ไหนว่างก็จะนำ thread นั้นไปทำงานหรือไปประมวลผล ส่วนแบบ critical จะเป็นการทยอยทำงาน ประมวลผล ซึ่งจะทำให้ใช้หลาย thread ในการทำงานซึ่งจะทำให้ประมวลผลช้านั่นเองและสุดท้ายแบบ reduction นั้นเป็นการให้ thread เข้าคิวประมวลผลทำงานซึงจะทำงานทีละ thread นั่นเอง

ตอนที่ 3 Section

Serial

Section Serial.c | For loop Parallel with reduction.c | For loop Parallel with critical section.c | For loop Par

```c
#include <stdio.h>
#define NUMBER 100
double alpha(){
 int i; double a=0.0;
 for (i=0; i<NUMBER; i++)
    a+=(i-1)/(i+1);
 return a;
}
double beta()
{
 int i; double b=0.0;
 for (i=NUMBER; i>0; i--)
 b+=i*(i-1);
 return b;
}
double delta(){
 int i,j; double c=0.0;
 for (i=NUMBER,j=0; i>0,i<NUMBER; i--,j++)
    c+=(i-j)/NUMBER;
 return c;
}
double gamma(double d, double e){
 return (d-e);
}
double epsilon(double f, double g){
 return (f+g);
}
int main(int argc, char *argv[]){
 double w, v, x, y, start,stop;
 start = omp_get_wtime();;
 w = alpha();
 v = beta();
 y = delta();
 x = gamma(v, w);
 printf ("%12.4f\n", x);
 printf ("%12.4f\n", epsilon(x,y));
 stop = omp_get_wtime();
 printf("Time to compile this = %f",stop-start);
}
```

ผลการ RUN

#define NUMBER 1000

G:\2560\lab advance I\Lab3\Section Serial.exe

```
 333301.0000
 333301.0000
Time to compile this = 1.000000
--------------------------------
Process exited after 0.01207 seconds with return value 31
Press any key to continue . . .
```

G:\2560\lab advance I\Lab3\Section Serial.exe

```
333333001.0000
333333001.0000
Time to compile this = 0.000000
--------------------------------
Process exited after 0.008369 seconds with return value 31
Press any key to continue . . .
```

Parallel

Section Parallel.c | Section Serial.c | For loop Parallel with reduction.c | For loop Parallel with critical section.c | F

```c
#include <omp.h>
#include <stdio.h>
#define NUMBER 100
double alpha(){
 int i; double a=0.0;
 for (i=0; i<NUMBER; i++)
 a+=(i-1)/(i+1);
 return a;
 }
double beta(){
 int i; double b=0.0;
 for (i=NUMBER; i>0; i--)
 b+=i*(i-1);
 return b;
 }
double delta(){
 int i,j; double c=0.0;
 for (i=NUMBER,j=0; i>0,i<NUMBER; i--,j++)
 c+=(i-j)/NUMBER;
 return c;
 }
double gamma(double d, double e){
 return (d-e);
}
double epsilon(double f, double g){
 return (f+g);
}
int main(int argc, char *argv[]){
 double start,stop;
 start = omp_get_wtime();
 double w, v, x, y;
 #pragma omp parallel
 {
 #pragma omp sections
 {
 #pragma omp section
 w = alpha();
 #pragma omp section
 v = beta();
 #pragma omp section
 y = delta();
 }
 }
 x = gamma(v, w);
 printf ("%12.4f\n", x); printf ("%12.4f\n", epsilon(x,y));
 stop = omp_get_wtime();
 printf("Time to compile this = %f",stop-start);
}
```

ผลการ RUN

```
G:\2560\lab advance I\Lab3\Section Parallel.exe
 333301.0000
 333301.0000
Time to compile this = 0.000000
--------------------------------
Process exited after 0.008882 seconds with return value 31
Press any key to continue . . .
```

```
G:\2560\lab advance I\Lab3\Section Parallel.exe
333333001.0000
333333001.0000
Time to compile this = 0.001000
--------------------------------
Process exited after 0.01055 seconds with return value 31
Press any key to continue . . .
```

#define NUMBER 1000

สรุปผลการทดลอง

ชนิดการประมวลผล	#define	Result	Time(s)
Serial	100	333301.0000	0.000000
Parallel	100	333301.0000	0.000000
Serial	1000	333333001.0000	0.001000
Parallel	1000	333333001.0000	0.001000

จากการทดลองจะพบว่า การปะมวลผลแบบ Serial ใน #define 1000 จะใช้เวลาการรันอยู่ที่ 0.001 และการประมวลผลแบบ Parallel ใน #define 1000 นั้นจะใช้เวลาเดียวกันซึ่งจริงๆในเวลาที่ทำที่แลปกับที่บ้านนั้นจะได้เวลาที่แตกต่างกันตามจริงแล้วเราจะพบว่าการประมวลผลแบบ Serial จะประมวลผลเร็วกว่า Parallel นั่นเอง

คำถามท้ายการทดลอง

1. จากตอนที่ 1 อธิบายว่าเหตุใดการแสดงผลลัพธ์ของโปรแกรมแบบขนานและอนุกรมจึงแตกต่างกัน

คำตอบ โปรแกรม Serial นั้นจะมีแค่ 1 thread ในการทำงานซึ่งจะเหมาะกับการทำงานที่ไม่ซับซ้อนหรือมีขั้นตอนการทำงานที่น้อยกว่า ส่วนโปรแกรมแบบขนาดนั้นจะมี thread มากกว่าหนึ่ง thread นั้นมันจะให้การทำงานนั้นเป็นขั้นเป็นตอนมากกว่า thread เดียวนั่นเองและเหมาะกับการทำงานของโปรแกรมที่มีขนาดใหญ่ หรือ ซับซ้อนเยอะๆนั่นเอง

2. จากตอนที่ 2 อธิบายว่าเหตุใดผลลัพธ์ที่ได้ของโปรแกรมในแบบขนานทั้งสามแบบไม่ตรงกัน

คำตอบ ในการทำงานแบบขนาดโปรแกรมจะทำงานโดยไม่มีการเรียงลำดับในการงานจึงทำให้ขั้นตอนในการประมวลผลนั้นมีข้อผิดพลาดเกิดขึ้นเยอะจึงทำให้โปรแกรมทั้ง 3 รูปแบบนั้นประมวลผลได้ไม่ตรงกันโดยแบบ navie นั้นจะเป็นการทำงานโดยดูว่า thread ไหนว่างก็จะนำ thread นั้นไปทำงานหรือไปประมวลผล ส่วนแบบ critical จะเป็นการทยอยทำงาน ประมวลผล ซึ่งจะทำให้ใช้หลาย thread ในการทำงานซึ่งจะทำให้ประมวลผลช้านั่นเองและสุดท้ายแบบ reduction นั้นเป็นการให้ thread เข้าคิวประมวลผลทำงานซึงจะทำงานทีละ thread นั่นเอง

3. จากตอนที่ 2 อธิบายว่าเหตุใดเวลาในการทำงานของโปรแกรมแบบ parallel ไม่เท่ากัน

คำตอบ เนื่องจากโปรแกรมแบบขนานนั้นจะมีการแบ่ง Thread การทำงานเป็นหลายๆ thread โดยแต่ละ thread ก็จะไปประมวลผลโดยทำงานของตัวเอง อย่างเช่น thread นี้มีหน้าที่ในการทำงานตรงนี้ ก็จะทำแค่ตรงนี้นั่นเอง

4. จากตอนที่ 3 อธิบายว่าเหตุใดเวลาในการทำงานของโปรแกรมแบบ parallel จึงไม่เร็วกว่า เวลาในการทำงานของโปรแกรมแบบ serial ทั้งที่แบ่ง section ให้ thread ช่วยกันทำงานแล้ว

คำตอบ การทำงานในแบบขนานนั้นจะใช้ thread หลาย thread มาช่วยในการทำงานซึ่งแตกต่างกับ Serial เพราะ Serial นั้นจะทำงานเพียง thread เดียวซึ่งทำงานเร็วกว่าหลาย thread แน่นอน